

# THE SUM-PRODUCT ALGORITHM FOR DEGREE-2 CHECK NODES AND TRAPPING SETS

JOHN O. BREVIK AND MICHAEL E. O'SULLIVAN

**ABSTRACT.** The sum-product algorithm for decoding of binary codes is analyzed for bipartite graphs in which the check nodes all have degree 2. The algorithm simplifies dramatically and may be expressed using linear algebra. Exact results about the convergence of the algorithm are derived and applied to trapping sets.

## 1. INTRODUCTION

One of the great achievements in coding theory in the last decade or so has been the discovery of iterative decoding methods, such as the sum-product algorithm (SPA). Experimental results on very long codes have yielded performance that is extremely close to Shannon capacity [11], and asymptotic analysis shows that ensembles of irregular codes achieve capacity [18]. Aside from the asymptotic theory, which is presented in detail in [19], there is little that has been proven about the performance of the sum-product algorithm. The girth and expansion coefficient of the associated bipartite graph, as well as low-weight pseudo-codewords and trapping sets (or near-codewords), are thought to affect the performance of the SPA, but we are not aware of any theorems that quantify the relationship for finite-length codes. In this article, we focus on a very special case for which we can derive exact results for convergence of the sum-product algorithm. By establishing some simple but provable results, we hope to build a foundation for further algebraic analysis.

We are solely concerned with binary codes. Fix such a code, defined as the right null space of a check matrix  $H$ . The SPA is most easily described via the bipartite graph of  $H$ , which has a check node for each row and a bit node for each column, with an edge between check node  $r$  and bit node  $\ell$  if and only if  $H_{r\ell} = 1$ . The initial data for the algorithm consists of a probability distribution for each bit, indicating the likelihoods that the transmitted signal for that bit is a 0 or a 1. The SPA then passes likelihood data along the edges of the bipartite graph from the check nodes to the bit nodes and back again. Since we wish to analyze the SPA, we will ignore the binary matrix defining the code, focusing on the equivalent description via the bipartite graph.

In this article we study bipartite graphs in which the check nodes all have degree 2. The SPA simplifies dramatically and may be analyzed using linear algebra. The codes defined by such graphs are repetition codes, provided the graph is connected, and therefore not of practical utility themselves. Nevertheless, there are surprising subtleties in the results. For example, our results indicate that in the

---

*Date:* January 4, 2011 version.

This research was supported by NSF CCF-Theoretical Foundations grants #0635382 and #0635389.

simple case of degree-2 checks, a covering graph can inherit provably bad convergence properties from its base graph. Furthermore, our results can be applied to the dynamics of the SPA in the presence of trapping sets, which have been studied extensively and identified as a powerful influence on decoding performance. In essence, we enhance a trapping set by adding some nodes that in some sense “virtually” supply messages from the rest of the graph; we then use our methods to study the SPA on this enhanced graph.

We are not aware of any previous work on this class of graphs, but, interestingly, there are several articles related to the opposite case, in which all bit nodes have degree 2. These codes are called *cycle codes* in [8, 9], which explore the connection to zeta functions of graphs and the fundamental cone of a parity-check matrix. Cycle codes were originally called *graph theoretic codes* or *circuit codes* in [3] and in previous work cited therein. Cycle codes from Ramanujan graphs are studied in [21].

The following section presents some necessary graph-theoretic terminology and the study of the *flow graph* arising from the SPA. Section 3 presents the sum-product algorithm and the simplifications due to having all checks of degree 2. Section 4 contains the two main theorems on convergence of the SPA. Section 5 presents several examples to illustrate the results. In Section 6 we analyze the SPA on trapping sets. In Section 7 we make several observations about the theoretical results and present some simulations of the SPA on small examples. These lead to questions for further investigation.

## 2. A DISCUSSION OF GRAPHS

The sum-product algorithm is defined via a bipartite graph, but our analysis of the algorithm will use two other graphs derived from the bipartite graph. First, a bipartite graph with all check nodes of degree 2 yields an undirected graph, and conversely. Second, the flow of information of the SPA suggests the construction of a graph whose vertices are the edges of the original bipartite graph. This is similar to the traditional notion of a *line graph* [5], but with some differences. In particular, our flow graph is a directed graph, containing one vertex for each *direction* of each edge in the bipartite graph. Finally, we also find it useful to allow graphs with loops and with multiple edges between a given pair of nodes.

In this section, we gather the formal definitions that we will use for bipartite graph, directed graph, and undirected graph; we present the constructions described above; and we establish some connectivity properties that will be used to analyze the SPA.

**Definition 2.1.** A *directed graph* is a 4-tuple  $(E, V, \sigma, \tau)$  consisting of a set  $E$  of edges, a set  $V$  of vertices (or nodes), and two maps  $\sigma : E \rightarrow V$  and  $\tau : E \rightarrow V$  giving the *source* and *terminus* of an edge.

The definition allows for  $\sigma(e) = \tau(e)$ , in which case  $e$  is a *loop*. The definition also allows for distinct edges  $e$  and  $f$  to have both the same source and the same terminus, which gives *parallel edges*.

**Definition 2.2.** An *undirected graph* is a directed graph with an involution  $E \rightarrow E : e \mapsto \bar{e}$  satisfying  $\bar{\bar{e}} = e$  and  $\tau(\bar{e}) = \sigma(e)$  for all  $e \in E$ .

As with a directed graph, loops and parallel edges are allowed. Note that with this definition, each edge depicted in the conventional drawing of an undirected

graph represents two conjugate edges. We will use double-headed arrows to emphasize this aspect of our conventions.

From a given directed graph  $G$ , there is an obvious way to obtain an undirected graph  $U(G)$ : One simply adds a set of conjugate edges,  $\overline{E} = \{\bar{e} \mid e \in E\}$ . The undirected graph  $U(G)$  has edge set  $E \cup \overline{E}$ , vertex set  $V$ , the obvious conjugation map and source and terminus maps extending  $\sigma$  and  $\tau$  to  $E \cup \overline{E}$  by  $\sigma(\bar{e}) = \tau(e)$  and  $\tau(\bar{e}) = \sigma(e)$ .

**Definition 2.3.** A *bipartite graph* is a directed graph along with a partition of the vertex set into two sets  $V_1, V_2$  such that every edge has source in  $V_1$  and terminus in  $V_2$ .

We will think of the edges going from “left” to “right,” so we will write a bipartite graph as a 5-tuple  $B = (E, L, R, \lambda, \rho)$ , in which  $L$  and  $R$  are the source nodes and terminus nodes, respectively. The maps  $\lambda : E \rightarrow L$  and  $\rho : E \rightarrow R$  give the source and terminus of an edge.

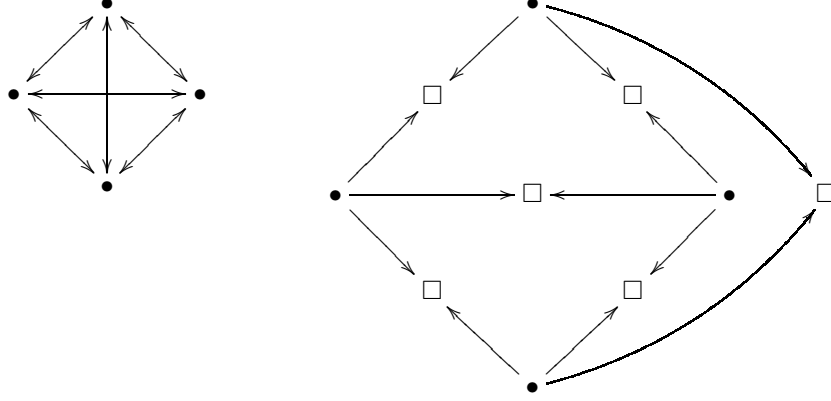
For the purpose of error-control coding, the elements of  $L$  are typically called *bit nodes* and the elements of  $R$  *check nodes*. A *codeword* is an association of 0 or 1 to each  $\ell \in L$  such that each  $r \in R$  is connected to an even number of nonzero bits. A binary matrix  $H$  yields a bipartite graph by taking  $R$  to be the set of rows of  $H$ ,  $L$  the set of columns of  $H$  and  $E$  enumerating the nonzero entries of  $H$ , so that for  $e$  the edge associated to the nonzero entry  $H_{r\ell}$ ,  $\lambda(e) = \ell$  and  $\rho(e) = r$ . A vector in the right nullspace of  $H$  gives a codeword for the associated graph. Note that while our definition allows for bipartite graphs that have parallel edges, such a graph clearly does not correspond to a binary matrix.

Given an undirected graph  $G = (E, V, \sigma, \tau)$ , we can form a bipartite graph  $(E, L, R, \lambda, \rho)$  in which all check nodes have degree 2 by putting a check node on each edge and making all edges point to the check nodes: Formally, take  $L = V$ ,  $\lambda = \sigma$ , and let  $R = E/\sim$  be the set  $E$  modulo the equivalence relation defined by the involution  $\smile$ ; now let  $\rho$  be the 2-to-1 map  $\rho : E \rightarrow R$ . The construction is illustrated in Figure 1.

Conversely, we may form an undirected graph from a bipartite graph  $B = (E, L, R, \lambda, \rho)$  that has all check nodes of degree 2 by removing each check node and treating the two edges meeting at a check node as a conjugate pair. Formally, for any edge  $e \in E$  let  $\bar{e}$  be the unique edge, distinct from  $e$ , sharing a node in  $R$  with  $e$ . Then let  $V = L$ ,  $\sigma = \lambda$  and let  $\tau(e) = \sigma(\bar{e})$ . We will say that this graph is *the undirected graph associated to  $B$* . It is clear that the two constructions are inverse operations. Note that the undirected graph associated to  $B$  is only defined when  $B$  has check nodes of degree 2, and it is not  $U(B)$ .

We will see in the next section that the SPA on a bipartite graph with checks of degree 2 simplifies to a linear algorithm, which we describe via the associated undirected graph  $G$ . The flow of information for the SPA follows paths in  $G$  that have no backtracking (no edge can follow its conjugate). Our analysis of the SPA will produce a matrix that is the adjacency matrix of a graph  $\tilde{G}$  derived from  $G$ . The rest of this section is devoted to the definition and analysis of  $\tilde{G}$ .

Let  $G = (V, E, \sigma, \tau)$  be a directed graph. We write a path in  $G$  as a sequence of edges  $e_1 \circ e_2 \circ \dots \circ e_n$  such that  $\tau(e_i) = \sigma(e_{i+1})$  for  $i = 1, \dots, n-1$ . We will say that  $\sigma(e_1)$  is connected to  $\tau(e_n)$  by this path. Notice that the relation “is connected to” is transitive, but not necessarily symmetric. Recall that  $G$  is *strongly connected* when any vertex  $v$  is connected to any other vertex  $w$ . For an undirected graph  $G$ ,

FIGURE 1.  $K_4$  and the bipartite graph associated to it.

we will say the path  $e_1 \circ e_2 \circ \cdots \circ e_n$  is *admissible* when the path does not involve any “backtracking,” that is,  $e_{i+1} \neq \bar{e}_i$  for  $i = 1, \dots, n-1$ .

The path  $e_1 \circ e_2 \circ \cdots \circ e_n$  is a cycle when  $\tau(e_n) = \sigma(e_1)$ . This cycle is *completely admissible* if it is admissible as a path and also  $e_n \neq \bar{e}_1$ . Thus a completely admissible cycle is one that is admissible when traversed starting at any of its vertices.

Let  $G$  be an undirected graph. The *flow graph* of  $G$  is the graph  $\tilde{G}$  with vertex set  $E$  and edge set  $\{(e, f) : \tau(e) = \sigma(f) \text{ and } \bar{f} \neq e\}$ . The source and terminus maps are projections:  $\tilde{\sigma}(e, f) = e$  and  $\tilde{\tau}(e, f) = f$ . There is a natural identification of paths in  $\tilde{G}$  with admissible paths in  $G$ . An admissible path  $e_1 \circ e_2 \circ \cdots \circ e_n$  of length  $n$  in  $G$  yields a path  $(e_1, e_2) \circ (e_2, e_3) \circ \cdots \circ (e_{n-2}, e_{n-1}) \circ (e_{n-1}, e_n)$  of length  $n-1$  in  $\tilde{G}$ , and conversely.

**Remark 2.4.** The flow graph is different from the line graph  $L(G)$  [5], since we include a vertex in  $\tilde{G}$  for each directed edge of  $G$ . It is also different from the usual line graph of a directed graph [1], since there is no edge in  $\tilde{G}$  between  $e$  and  $\bar{e}$ . Stark and Terras [20, §3] define an edge zeta function for  $G$  by constructing the *directed edge matrix* of  $G$ . Although they do not construct the flow graph, their directed edge matrix is the adjacency matrix of the flow graph.

**Proposition 2.5.** *Let  $G$  be an undirected graph, and let  $\tilde{G}$  be as defined above. There is a natural length-preserving bijection between cycles in  $\tilde{G}$  and completely admissible cycles in  $G$ .*

*Proof.* Let  $e_1 \circ e_2 \circ \cdots \circ e_n$  be a completely admissible cycle in  $G$ . Then since  $\tau(e_n) = \sigma(e_1)$ ,  $(e_1, e_2) \circ (e_2, e_3) \circ \cdots \circ (e_{n-2}, e_{n-1}) \circ (e_{n-1}, e_n) \circ (e_n, e_1)$  is a cycle in  $\tilde{G}$ , also of length  $n$ . Conversely, a cycle in  $\tilde{G}$  is easily seen to give a cycle in  $G$ , of the same length, which must be completely admissible.  $\square$

The two examples in Figures 2 and 3 show that  $\tilde{G}$  is not in general an undirected graph. The examples also show that  $\tilde{G}$  need not be strongly connected, even when  $G$  is. In Figure 2 the undirected graph  $U(\tilde{G})$  derived from  $\tilde{G}$  is not even connected. It is clear this phenomenon occurs when  $G$  is a path or cycle. In Figure 3,  $\tilde{G}$  is not

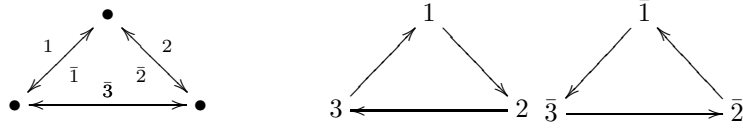


FIGURE 2. On the left, the graph  $G$  is an undirected 3-cycle. The outer labels are for clockwise edges, the inner labels for counterclockwise edges. On the right,  $\tilde{G}$  consists of two disconnected directed 3-cycles.

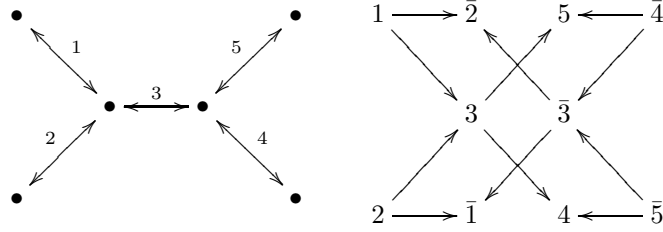


FIGURE 3. On the left, the graph  $G$  is an undirected tree. The edges 1,2,3,4,5 go left to right, and the conjugate edges go right to left. On the right the flow graph,  $\tilde{G}$  is not strongly connected (note, *e.g.*, that node 1 has no edges leading into it), although  $U(\tilde{G})$  is connected.

strongly connected, since no edge in  $\tilde{G}$  has source 5, but  $U(\tilde{G})$  is connected. The following propositions present some connectivity properties of  $\tilde{G}$ .

**Lemma 2.6.** *Let  $G$  be a connected undirected graph. Let  $e, f$  be edges with  $f \neq e$ , and  $f \neq \bar{e}$ . In  $\tilde{G}$ , either  $e$  or  $\bar{e}$  is connected either to  $f$  or to  $\bar{f}$  by an admissible path. Consequently,  $U(\tilde{G})$  has at most two connected components.*

*If  $G$  has a vertex of degree at least 3, then  $U(\tilde{G})$  is connected.*

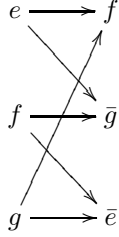
*Proof.* Let  $v = \tau(e), w = \sigma(f)$ . If  $v = w$ , then there is an edge  $(e, f)$  in  $\tilde{G}$  connecting  $e$  and  $f$ , since we assume  $f \neq \bar{e}$ . If  $v \neq w$ , then, since  $G$  is connected, there is a path  $P$  from  $v$  to  $w$ , and we can eliminate any backtracking to make  $P$  admissible.

If  $P$  begins with  $\bar{e}$  and ends with  $\bar{f}$ , then  $\bar{e}$  is connected to  $\bar{f}$ . If  $P$  begins with  $\bar{e}$  and does not end with  $\bar{f}$ , then  $P \circ \bar{f}$  is admissible and shows that  $\bar{e}$  is connected to  $\bar{f}$ .

If  $P$  does not begin with  $\bar{e}$ , then the path  $e \circ P$  is still admissible; now proceed as in the preceding paragraph.

Notice that an admissible path from  $\bar{e}$  to  $f$  yields, by reversing direction, an admissible path from  $\bar{f}$  to  $e$ . We have shown, therefore, that in any conjugate pair  $\{f, \bar{f}\}$  either  $e$  is connected to one of the edges, or one of the edges is connected to  $e$ . Consequently,  $U(\tilde{G})$  has at most two connected components, one containing the edges connected by an admissible path to (or from)  $e$ , and the other those connected by an admissible path to (or from)  $\bar{e}$ .

Suppose  $G$  has a vertex  $z$  of degree at least 3, and let  $e, f, g$  be three distinct edges with terminus  $z$ . Then  $\tilde{G}$  has the subgraph



This shows that  $e$  and  $\bar{e}$  are in the same connected component of  $U(\tilde{G})$ . By transitivity,  $U(\tilde{G})$  is connected.  $\square$

It is clear that  $\tilde{G}$  is not strongly connected when  $G$  has a vertex of degree one. The major result of this section, Proposition 2.10, says that when  $G$  is connected, has no vertices of degree 1, and has one vertex of degree at least 3,  $\tilde{G}$  is strongly connected.

**Lemma 2.7.** *Let  $G$  be an undirected graph such that  $G$  is connected, every vertex has degree  $\geq 2$ , and some vertex has degree  $\geq 3$ . Then*

- (1) *Every cycle on  $G$  contains a vertex of degree  $\geq 3$ .*
- (2) *Every edge on  $G$  is connected via an admissible path, possibly empty, to an admissible cycle in such a way that neither the path nor its conjugate has an edge in common with the cycle.*

*Proof.* Suppose  $C$  is a cycle on  $G$  all of whose vertices have degree 2.  $C$  is clearly a connected component of  $G$ , therefore all of  $G$  itself since  $G$  is connected; this contradicts the assumption that  $G$  has a vertex of degree  $\geq 3$ . This establishes the first item in the theorem.

Let  $e$  be a given edge; follow an admissible path from  $e$ , choosing edges arbitrarily. Since every vertex of  $G$  has degree  $\geq 2$ , the path can be extended admissibly at every step until it reaches for the first time a vertex  $w$  already visited. The path from  $w$  to  $w$  is necessarily an admissible cycle. Edges in this cycle,  $C$ , share at most one vertex with edges of the path  $P$  from  $\sigma(e)$  to  $w$ . Thus there is no edge common to  $C$  and either  $P$  or  $\bar{P}$ .  $\square$

**Lemma 2.8.** *Let  $G$  be as in Lemma 2.7. Then every edge on  $G$  is connected via an admissible path to its opposite.*

*Proof.* Let  $e$  be a given edge. By Lemma 2.7,  $e$  is connected via an admissible path  $P$  to an admissible cycle  $C$ . If  $P$  is nonempty, then  $P \circ C \circ \bar{P}$  connects  $e$  to  $\bar{e}$  admissibly.

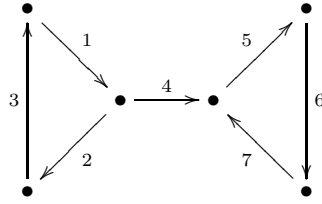
If  $P$  is empty, then  $e$  lies on  $C$ . Starting at  $v = \sigma(e)$ , follow  $C$  along a nonempty path  $S$  (possibly equal to  $C$  itself) to a vertex  $w$  of degree  $\geq 3$ . Then  $w$  is the source of an edge  $f$  not on  $C$  or  $\bar{C}$ . By Lemma 2.7,  $f$  is connected by a path  $Q$  to a simple cycle  $D$ .

- Case 1: Suppose  $Q$  is nonempty, so  $f$  is not part of  $D$ . Then  $Q \circ D \circ \bar{Q}$  connects  $f$  to  $\bar{f}$  and  $S \circ Q \circ D \circ \bar{Q} \circ \bar{S}$  is an admissible path from  $e$  to  $\bar{e}$ .

- Case 2: Suppose  $Q$  is empty, so  $f$  lies on  $D$ . Let  $z$  be the first vertex (beyond the initial  $w$ ) lying on  $D \cap C$ , and let  $T$  be the corresponding path from  $w$  to  $z$  ( $z$  may equal  $w$ ). Note that  $S \circ T$  is admissible, since the first edge of  $T$  is  $f$  which does not lie on  $C$ . Now let  $U$  be the segment of  $\bar{C}$  from  $z$  back to  $v$ . Since the final edge of  $T$  is not on  $C$ ,  $T \circ U$  is admissible, and therefore so is  $S \circ T \circ U$ , connecting  $e$  to  $\bar{e}$ .

□

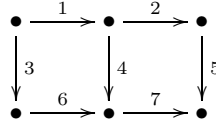
**Examples 2.9.** The following graphs illustrate the two cases in the above proof. First consider the undirected graph  $G$  derived by adding conjugate edges to the following graph:



Note that edge 1 is connected to edge  $\bar{1}$  via the edge sequence  $1 \circ 4 \circ \underbrace{5 \circ 6 \circ 7}_D \circ \bar{4} \circ \bar{1}$ .

This is the path constructed in Case 1 of the above proof, in which  $\bar{f}$  is edge 4, path  $S$  is edge 1, and path  $T$  is edge 4.

For Case 2, which  $f$  does lie on the cycle,  $D$ , consider the undirected graph derived from



To connect edge 3 to edge  $\bar{3}$ , view edge 3 as lying on the cycle  $C : 3 \circ 6 \circ \bar{4} \circ \bar{1}$ , and follow  $\underbrace{3 \circ 6}_S \circ \underbrace{7 \circ 5 \circ 2}_T \circ \underbrace{\bar{4} \circ \bar{6} \circ \bar{3}}_U$ . Here of course  $f = 7$  lies on  $D : 7 \circ 5 \circ 2 \circ \bar{1} \circ 3 \circ 6$ .

Note that it was important to rejoin the cycle  $C$  at the first opportunity, namely at edge 4; this is the only juncture that allows us to reverse directions on  $C$ .

**Proposition 2.10.** *Let  $G$  be an undirected graph such that  $G$  is connected, every vertex has degree  $\geq 2$ , and some vertex has degree  $\geq 3$ . Then for any two edges  $e, f$  in  $G$ ,  $e$  is connected to  $f$  via an admissible path. Consequently,  $\bar{G}$  is strongly connected.*

*Proof.* By Lemma 2.6, there is an admissible path  $P$  connecting either  $e$  or  $\bar{e}$  either to  $f$  or to  $\bar{f}$ . If  $P$  connects  $e$  to  $f$ , there is nothing to prove.

If  $P$  connects  $e$  to  $\bar{f}$ , use Lemma 2.8 to connect  $\bar{f}$  admissibly to  $f$  via the admissible path  $Q = \bar{f} \circ Q'$ . Now the concatenation  $P \circ Q'$  is admissible, since the first edge of  $Q'$  cannot be  $f$  and so is not the reverse of the last edge  $\bar{f}$  of  $P$ , and  $P \circ Q'$  connects  $e$  to  $f$ .

If  $P$  connects  $\bar{e}$  to  $f$ , use Lemma 2.8 to connect  $e$  to  $\bar{e}$  by an admissible path  $R = R' \circ \bar{e}$ ; as in the previous paragraph,  $R' \circ P$  is admissible and connects  $e$  to  $f$ .

If  $P$  connects  $\bar{e}$  to  $\bar{f}$ , use  $R' \circ P \circ Q'$  as constructed in the preceding paragraphs.

□

### 3. THE SUM-PRODUCT ALGORITHM WITH CHECKS OF DEGREE 2

Throughout this section and the next, let  $B = (E, L, R, \lambda, \rho)$  be a bipartite graph. In this section we show how the sum-product algorithm simplifies when all check nodes have degree 2.

We express all of the probabilistic data in the sum-product algorithm using the *odds ratio*, which for a distribution  $p$  on  $\{0, 1\}$  is  $p(1)/p(0)$ . The input to the sum-product algorithm is then  $u_\ell = p_\ell(1)/p_\ell(0)$  where  $p_\ell$  expresses the likelihood, given some received signal for bit  $\ell$ , that this bit's value is either 1 or 0. Likewise, the messages along the edges of the graph produced by the algorithm are expressed as the odds of 1. We define the *parity* of  $u \in (0, \infty)$  to be 0 if  $u < 1$ ,  $\infty$  if  $u > 1$ , and undefined when  $u = 1$ . The sum-product algorithm uses the transform from the “odds of 1” domain to the “difference domain” in which a probability distribution  $p$  is represented using  $p(0) - p(1)$ , which is in the interval  $[-1, +1]$ . The function  $s : \mathbb{R} \cup \{\infty\} \rightarrow \mathbb{R} \cup \{\infty\}$  defined by  $s(x) = \frac{1-x}{1+x}$  transforms from one domain to the other. Notice that  $s(s(x)) = x$ .

**Algorithm 3.1** (Sum-Product Algorithm).

INPUT: For each  $\ell \in L$ ,  $u_\ell \in (0, \infty)$ . Termination criterion  $\epsilon > 0$ .

DATA STRUCTURES: For each  $e \in E$ ,  $x_e, y_e \in (0, \infty)$ .

INITIALIZATION: Set  $y_e \leftarrow 1$  for all  $e \in E$ .

ALGORITHM:

BIT-TO-CHECK STEP: For each  $e \in E$ , set

$$x_e \leftarrow u_{\lambda(e)} \prod_{\substack{f: \lambda(f)=\lambda(e) \\ f \neq e}} y_f$$

CHECK-TO-BIT STEP: For each  $e \in E$ , set

$$y_e \leftarrow s \left( \prod_{\substack{f: \rho(f)=\rho(e) \\ f \neq e}} s(x_f) \right)$$

NEW ESTIMATE STEP: Set

$$\hat{u}_\ell \leftarrow u_\ell \prod_{e \in \lambda^{-1}(\ell)} y_e$$

TERMINATION AND OUTPUT: If either  $\hat{u}_\ell < \epsilon$  or  $\hat{u}_\ell > 1/\epsilon$  for all  $\ell \in L$  then return the binary vector based on the parity of  $\hat{u}_\ell$ : Vector  $w \in \mathbb{F}^L$  such that

$$w_\ell = \begin{cases} 1 & \text{if } \hat{u}_\ell > 1 \\ 0 & \text{else} \end{cases}$$

There are a variety of reasonable criteria for termination; we have simply chosen one. We are interested in finding conditions on the set of  $u_\ell$  that will determine the convergence behavior of the set of  $\hat{u}_\ell$ .



When analyzing the algorithm it will sometimes prove useful to indicate the iteration using a superscript as follows. We initialize  $y^{(0)} = 1$  and for  $t \geq 1$ ,

$$x_e^{(t)} \leftarrow u_{\lambda(e)} \prod_{\substack{f:\lambda(f)=\lambda(e) \\ f \neq e}} y_f^{(t-1)} \quad \text{and} \quad y_e^{(t)} \leftarrow s \left( \prod_{\substack{f:\rho(f)=\rho(e) \\ f \neq e}} s(x_f^{(t)}) \right)$$

As we have defined it, a bipartite graph is *directed*; all edges go from bit nodes to check nodes. One could also describe the algorithm using the undirected graph  $U(B)$  discussed in the previous section. The messages  $y_e$  may then be considered as attached to the reverse arrow  $\bar{e}$ . The notation proved to be simpler using the directed bipartite graph since we will define a conjugation map on  $E$  itself when all check nodes have degree 2.

We now restrict attention to bipartite graphs in which each check node has degree 2. We also assume that the graph is connected, since the SPA treats each component independently. One may readily check that the code defined by such a graph is a repetition code. The sum-product algorithm simplifies dramatically because at the check to bit step there is only one term in the product.

**Proposition 3.2.** *If all check nodes have degree 2, then all edge messages are monomials in the  $u_\ell$ . Furthermore, for each edge  $e$ , at any iteration  $t$ ,  $y_e^{(t)} = x_{\bar{e}}^{(t)}$  where  $\bar{e}$  is the unique edge sharing a check node with  $e$ .*

*Proof.* Clearly, at initialization  $y_e^{(0)} = 1$  is a monomial, as claimed. Moreover, if all  $y_e^{(t)}$  are monomials, then all  $x_e^{(t+1)}$  are monomials as well, since the bit-to-check step just involves multiplication. Each right node has degree 2, so the product in the check-to-bit step has only one term. Since  $s^2$  is the identity,  $y_e^{(t+1)} = x_{\bar{e}}^{(t+1)}$  where  $\bar{e}$  is the unique edge distinct from  $e$  sharing the same right node. Thus we may establish the proposition by induction.  $\square$

It is now evident that, when all check nodes have degree 2, the check nodes play no significant role in the algorithm, and that analysis of the algorithm is simplified by keeping track only of the exponents for the monomials  $x_e^{(t)}$ . We can express the SPA using the undirected graph  $G$  associated to  $(E, L, R, \lambda, \rho)$  (cf. §2). Let us use  $\mathbf{a}_e \in \mathbb{N}^L$  to denote the row vector of exponents appearing in  $x_e$ , so  $x_e = \prod_{\ell \in L} u_\ell^{a_{e,\ell}}$ . We will abbreviate this product as  $\mathbf{u}^{\mathbf{a}_e}$ . When we want to specify the  $t^{\text{th}}$  iteration we will write  $\mathbf{a}_e^{(t)}$ . Let  $\mathbf{0} \in \mathbb{N}^L$  be the all-0 row vector and let  $\delta_\ell \in \mathbb{N}^L$  be the row vector which is 1 in the  $\ell^{\text{th}}$  component and 0 otherwise. The update in the SPA is  $x_e \leftarrow u_{\lambda(e)} \prod_{\substack{f:\lambda(f)=\lambda(e) \\ f \neq e}} x_{\bar{f}}$ , or in terms of  $\mathbf{u}$  and  $\mathbf{a}_e$ ,

$$x_e = \mathbf{u}^{\mathbf{a}_e} \leftarrow u_{\lambda(e)} \prod_{\substack{f:\lambda(\bar{f})=\lambda(e) \\ \bar{f} \neq e}} \mathbf{u}^{\mathbf{a}_f}$$

Keeping track of the exponents gives us the following algorithm.

**Algorithm 3.3** (Local Sum Algorithm).

DATA STRUCTURES: For each  $e \in E$ ,  $\mathbf{a}_e \in \mathbb{N}^L$ .

INITIALIZATION: Set  $\mathbf{a}_e \leftarrow \mathbf{0}$  for all  $e \in E$ .

ALGORITHM: Set

$$(1) \quad \mathbf{a}_e \leftarrow \delta_{\lambda(e)} + \sum_{\substack{f: \lambda(\bar{f}) = \lambda(e) \\ \bar{f} \neq e}} \mathbf{a}_f$$

Let  $\mathbf{A}$  be the  $|E| \times |L|$  matrix whose  $e^{\text{th}}$  row is  $\mathbf{a}_e$ . Let  $\mathbf{\Lambda}$  be the  $|E| \times |L|$  matrix and let  $\mathbf{K}$  be the  $|E| \times |E|$  matrix defined by

$$\mathbf{\Lambda}_{e,l} = \begin{cases} 1 & \text{when } \lambda(e) = l \\ 0 & \text{else} \end{cases} \quad \mathbf{K}_{e,f} = \begin{cases} 1 & \text{when } \lambda(\bar{f}) = \lambda(e) \text{ and } \bar{f} \neq e \\ 0 & \text{else} \end{cases}$$

The  $e^{\text{th}}$  row of  $\mathbf{\Lambda}$  is  $\delta_{\lambda(e)}$  and one can check that  $\mathbf{K}$  is the adjacency matrix of the graph  $\tilde{G}$ .

The local sum algorithm is then

$$\begin{aligned} \mathbf{A}^{(0)} &= \mathbf{0} \\ \mathbf{A}^{(t)} &= \mathbf{\Lambda} + \mathbf{K}\mathbf{A}^{(t-1)} \end{aligned}$$

The equation above is easily solved, for  $t \geq 1$ ,

$$\mathbf{A}^{(t)} = (\mathbf{K}^{t-1} + \mathbf{K}^{t-2} + \cdots + \mathbf{K} + \mathbf{I})\mathbf{\Lambda}$$

#### 4. CONVERGENCE OF THE SPA

Throughout this section we adopt the following notation. Let  $B = (E, L, R, \lambda, \rho)$  be a bipartite graph. We assume that  $U(B)$  is connected and that

- all check nodes have degree 2,
- all bit nodes have degree at least 2,
- some bit node has degree at least 3.

Let  $G$  be the associated undirected graph as constructed in Section 2 and let  $\tilde{G}$  be the flow graph of  $G$ . Let  $\mathbf{K}$  be the adjacency matrix of  $\tilde{G}$ , which is the matrix of the local sum algorithm.

A vector or matrix is said to be *nonnegative* if each of its entries is nonnegative, so  $K$  is nonnegative. Similarly, a vector or matrix is *positive* when each entry is positive. We first treat the case in which  $\mathbf{K}$  is *primitive*, that is,  $\mathbf{K}^r$  is positive for some positive integer  $r$ . By the Perron-Frobenius theorem [6, 8.2.5], [13, Ch. 1]  $\mathbf{K}$  has a real positive eigenvalue  $\rho$  satisfying the following.

- $\rho$  is between the maximum row sum and the minimum row sum of  $\mathbf{K}$ , and strictly between the two if the maximum and minimum are not equal.
- $\rho$  has algebraic multiplicity 1.
- $\rho$  is strictly larger in modulus than all other eigenvalues of  $\mathbf{K}$ .
- The eigenvectors associated to  $\rho$  are strictly positive.
- Let  $\mathbf{z}$  be a right eigenvector and  $\mathbf{y}^*$  a left eigenvector associated to  $\rho$ , normalized so that  $\mathbf{y}^*\mathbf{z} = 1$ . Then [6, 8.2.7]  $\mathbf{K}^i = \rho^i\mathbf{z}\mathbf{y}^* + (\mathbf{K} - \rho\mathbf{z}\mathbf{y}^*)^i$  and the eigenvalues of  $(\mathbf{K} - \rho\mathbf{z}\mathbf{y}^*)^i$  have modulus less than  $\rho$ .

The eigenvectors  $\mathbf{y}^*$  and  $\mathbf{z}$  are called *Perron vectors* of  $\mathbf{K}$ . As a consequence of the final point, for large powers of  $i$ ,  $\mathbf{K}^i$  is approximated by  $\rho^i\mathbf{z}\mathbf{y}^*$ .

**Theorem 4.1.** *With the notation above, let  $\mathbf{c} = \mathbf{y}^*\mathbf{\Lambda}$ . The sum-product algorithm on  $B$  converges based on the parity of  $\mathbf{u}^{\mathbf{c}}$ . That is, the algorithm converges to 0 when  $\mathbf{u}^{\mathbf{c}} < 1$  and to  $\infty$  when  $\mathbf{u}^{\mathbf{c}} > 1$ .*

*Proof.* Since we assumed that the bipartite graph  $B$  has all vertices of degree at least 2 and one vertex of larger degree, the row sums of  $\mathbf{K}$  are at least 1 and strictly greater than 1 for some edge. Thus  $\rho > 1$ . We have  $\mathbf{K}^i = \rho^i \mathbf{z}\mathbf{y}^* + (\mathbf{K} - \rho \mathbf{z}\mathbf{y}^*)^i$  and therefore

$$\begin{aligned} \sum_{i=0}^{t-1} \mathbf{K}^i &= \frac{\rho^t - 1}{\rho - 1} \mathbf{z}\mathbf{y}^* + \sum_{i=0}^{t-1} (\mathbf{K} - \rho \mathbf{z}\mathbf{y}^*)^i \\ \frac{\rho - 1}{\rho^t - 1} \sum_{i=0}^{t-1} \mathbf{K}^i &= \mathbf{z}\mathbf{y}^* + \frac{\rho - 1}{\rho^t - 1} \sum_{i=0}^{t-1} (\mathbf{K} - \rho \mathbf{z}\mathbf{y}^*)^i \end{aligned}$$

Since the eigenvalues of  $(\mathbf{K} - \rho \mathbf{z}\mathbf{y}^*)$  are less than  $\rho$  in modulus, the final term in the last equation goes to 0 as  $t$  goes to  $\infty$ . Thus

$$\begin{aligned} \lim_{t \rightarrow \infty} \frac{\rho - 1}{\rho^t - 1} \mathbf{A}^{(t)} &= \mathbf{z}\mathbf{y}^* \mathbf{\Lambda} \\ \lim_{t \rightarrow \infty} \frac{\rho - 1}{\rho^t - 1} \mathbf{a}_e^{(t)} &= \mathbf{z}_e \mathbf{y}^* \mathbf{\Lambda} \\ \lim_{t \rightarrow \infty} \left( x_e^{(t)} \right)^{\frac{\rho-1}{\rho^t-1}} &= \lim_{t \rightarrow \infty} (\mathbf{u} \mathbf{a}_e^{(t)})^{\frac{\rho-1}{\rho^t-1}} \\ &= (\mathbf{u}^{\mathbf{c}})^{\mathbf{z}_e} \end{aligned}$$

where  $\mathbf{c} = \mathbf{y}^* \mathbf{\Lambda}$ . Since  $\rho > 1$ ,  $0 < \frac{\rho-1}{\rho^t-1} < 1$ . Thus for any edge  $e$ ,  $x_e$  goes to 0 if  $\mathbf{u}^{\mathbf{c}} < 1$  and to  $\infty$  if  $\mathbf{u}^{\mathbf{c}} > 1$ .  $\square$

The proof gives information about the rate of convergence, which, for all edges, is roughly exponential with exponent  $\rho$ ,  $x_e^{(t)} \approx ((\mathbf{u}^{\mathbf{c}})^{\mathbf{z}_e})^{\frac{\rho^t-1}{\rho-1}}$ . The base for the exponential growth,  $(\mathbf{u}^{\mathbf{c}})^{\mathbf{z}_e}$ , depends upon the edge. The following corollary summarizes this result, and the analagous statement for the rate of convergence of the new estimates,  $\hat{u}_\ell$ , which varies with  $\ell$ .

**Corollary 4.2.** *The limiting behavior of  $x_e$  and  $\hat{u}_\ell$  at iteration  $t$  are as follows.*

$$\begin{aligned} \lim_{t \rightarrow \infty} \left( x_e^{(t)} \right)^{\frac{\rho-1}{\rho^t-1}} &= (\mathbf{u}^{\mathbf{c}})^{\mathbf{z}_e} \\ \lim_{t \rightarrow \infty} \left( \hat{u}_\ell^{(t)} \right)^{\frac{\rho-1}{\rho^t-1}} &= (\mathbf{u}^{\mathbf{c}})^{\sum_{e \in \lambda^{-1}(\ell)} \mathbf{z}_e} \end{aligned}$$

Let us now turn to the general case, in which  $\mathbf{K}$  may not be primitive. Our assumptions on the bipartite graph  $B$  ensure that  $\tilde{G}$  is strongly connected. This is equivalent to  $\mathbf{K}$  being *irreducible* [6, Thm. 6.2.24] [13, Ch. 4, Thm. 3.2], since it is the adjacency matrix of  $\tilde{G}$ . Thus we are led to the theory of irreducible matrices.

A square matrix  $H$  is *reducible* when there exists a permutation matrix  $P$  such that  $P^\dagger H P = \begin{bmatrix} A & B \\ 0 & C \end{bmatrix}$  with  $A$  and  $C$  square matrices. When  $H$  is not reducible, it is called *irreducible*. If  $H$  is the adjacency matrix of a graph, then it is straightforward to show that  $H$  is reducible if and only if there is a nontrivial partition of the vertex set  $V$  into  $V_1, V_2$  such that there is no edge from  $V_2$  to  $V_1$ . There are several other equivalent conditions for irreducibility in [6, §6.2] and in [13, §1.2].

The important properties for analysis of the sum-product algorithm appear in [13, Ch.3, Ch. 4 §3]. Since  $\mathbf{K}$  is irreducible, it has a positive eigenvalue  $\rho$  of

maximum modulus. There is a positive integer  $h$ , called the *index of imprimitivity* of  $\mathbf{K}$ , satisfying the following equivalent conditions.

- $\mathbf{K}$  has  $h$  eigenvalues of modulus  $\rho$ .
- $h$  is the largest positive integer such that  $\mathbf{K}^h$  is a block-diagonal matrix with  $h$  blocks, each an irreducible matrix.
- $h$  is the largest integer for which there is a partition of  $E$  into disjoint sets  $E_1, \dots, E_h$  such that any edge of  $\tilde{G}$  goes from  $E_{i+1}$  to  $E_i$  or  $E_1$  to  $E_h$ .
- The greatest common divisor of the lengths of the cycles in  $\tilde{G}$  is  $h$ .

Much more is known. The  $h$  eigenvalues of  $\mathbf{K}$  with modulus  $\rho$  are  $\rho\zeta^i$  where  $\zeta$  is an  $h^{\text{th}}$  root of unity. Enumerating the elements of  $E$  by first taking the elements of  $E_1$ , then  $E_2$ , and continuing on to  $E_h$ , the matrix  $\mathbf{K}$  has the form

$$\mathbf{K} = \begin{bmatrix} \mathbf{0} & \mathbf{K}_1 & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{K}_2 & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{K}_3 & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{K}_{h-1} \\ \mathbf{K}_h & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \end{bmatrix}$$

with square  $\mathbf{0}$  matrices along the diagonal. Computing  $\mathbf{K}^h$ , one can see that it is block diagonal with  $j^{\text{th}}$  block equal to  $\mathbf{K}_j \mathbf{K}_{j+1} \cdots \mathbf{K}_h \mathbf{K}_1 \cdots \mathbf{K}_{j-1}$ . The conditions above state that this product is irreducible for all  $i$ , but the maximality of  $h$  actually ensures that these products are all primitive. Furthermore, they have the same nonzero eigenvalues, since for any matrices  $A, B$  with compatible dimensions the nonzero eigenvalues of  $AB$  and  $BA$  are the same. Finally, we note that Proposition 2.5 shows that the gcd of the cycle lengths in  $\tilde{G}$  equals the gcd of the lengths of completely admissible cycles in  $G$ .

Let  $\mathbf{y}_1^*$  and  $\mathbf{z}_1$  be Perron vectors for the product  $\mathbf{K}_1 \mathbf{K}_2 \cdots \mathbf{K}_h$  satisfying  $\mathbf{y}_1^* \mathbf{z}_1 = 1$ . For  $j = 1, \dots, h$ , define

$$\begin{aligned} \mathbf{y}_j^* &= \rho^{1-j} \mathbf{y}_1^* \mathbf{K}_1 \mathbf{K}_2 \cdots \mathbf{K}_{j-1} \\ \mathbf{z}_j &= \rho^{j-h-1} \mathbf{K}_j \mathbf{K}_{j+1} \cdots \mathbf{K}_h \mathbf{z}_1 \end{aligned}$$

Now,  $\mathbf{y}_j^*$  and  $\mathbf{z}_j$  are Perron vectors for  $\mathbf{K}_j \mathbf{K}_{j+1} \cdots \mathbf{K}_h \mathbf{K}_1 \cdots \mathbf{K}_{j-1}$ , as is readily verified; moreover,

$$\begin{aligned} \mathbf{y}_j^* \mathbf{z}_j &= \rho^{-h} \mathbf{y}_1^* \mathbf{K}_1 \mathbf{K}_2 \cdots \mathbf{K}_h \mathbf{z}_1 = 1 \\ \mathbf{y}_j^* \mathbf{K}_j &= \rho \mathbf{y}_{j+1}^* \\ \mathbf{K}_j \mathbf{z}_{j+1} &= \rho \mathbf{z}_j \end{aligned}$$

where the subscripts are computed modulo  $h$ , with representatives  $\{1, 2, \dots, h\}$ .

Note that  $\mathbf{z}_j$  is a column vector with  $|E_j|$  entries and  $\mathbf{y}_j^*$  is a row vector with  $|E_j|$  entries. We form the  $|E| \times h$  matrix  $\mathbf{Z}$  and the  $h \times |E|$  matrix  $\mathbf{Y}$  as follows:

$$\mathbf{Z} = \begin{bmatrix} \mathbf{z}_1 & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{z}_2 & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{z}_3 & \dots & \mathbf{0} \\ \dots & \dots & \dots & \dots & \dots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{z}_h \end{bmatrix}, \quad \mathbf{Y} = \begin{bmatrix} \mathbf{y}_1^* & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{y}_2^* & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{y}_3^* & \dots & \mathbf{0} \\ \dots & \dots & \dots & \dots & \dots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{y}_h^* \end{bmatrix}$$

We also form the  $h \times h$  matrices

$$\Theta = \begin{bmatrix} 1 & \rho & \rho^2 & \dots & \rho^{h-1} \\ \rho^{h-1} & 1 & \rho & \dots & \rho^{h-2} \\ \rho^{h-2} & \rho^{h-1} & 1 & \dots & \rho^{h-3} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \rho & \rho^2 & \rho^3 & \dots & 1 \end{bmatrix}, \quad \mathbf{P} = \begin{bmatrix} 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & 0 & 0 & \dots & 0 \end{bmatrix}$$

**Lemma 4.3.** *With the matrices  $\mathbf{Z}, \mathbf{Y}, \Theta, \mathbf{P}$  defined above and for any  $0 \leq r < h$ ,*

$$\lim_{q \rightarrow \infty} \frac{\rho^h - 1}{\rho^{hq} - 1} \sum_{i=0}^{hq+r-1} \mathbf{K}^i = \rho^r \mathbf{Z} \mathbf{P}^r \Theta \mathbf{Y}$$

*Proof.* First we write

$$\begin{aligned} \sum_{i=0}^{hq+r-1} \mathbf{K}^i &= \mathbf{K}^r \left( \sum_{i=0}^{hq-1} \mathbf{K}^i \right) + \left( \mathbf{K}^{r-1} + \dots + \mathbf{I} \right) \\ &= \mathbf{K}^r \left( \sum_{j=0}^{q-1} \mathbf{K}^{hj} \right) \left( \mathbf{K}^{h-1} + \mathbf{K}^{h-2} + \dots + \mathbf{K} + \mathbf{I} \right) + \left( \mathbf{K}^{r-1} + \dots + \mathbf{I} \right) \end{aligned}$$

We may ignore the last term since multiplying it by  $\frac{\rho^h - 1}{\rho^{hq} - 1}$  and taking the limit as  $q$  goes to infinity gives 0. Recall that  $\mathbf{K}^h$  is a block-diagonal matrix with entries  $\mathbf{K}_1 \dots \mathbf{K}_h$ ,  $\mathbf{K}_2 \dots \mathbf{K}_h \mathbf{K}_1$ , and so on to  $\mathbf{K}_h \mathbf{K}_1 \dots \mathbf{K}_2$ . Each of these matrices is primitive and each has  $\rho^h$  as its largest eigenvalue. As in the proof of Theorem 4.1, where  $\mathbf{K}$  is primitive, large powers of these matrices are approximated using the product of Perron vectors. Thus we have

$$\lim_{q \rightarrow \infty} \frac{\rho^h - 1}{\rho^{hq} - 1} \sum_{j=0}^{q-1} \mathbf{K}^{hj} = \begin{bmatrix} \mathbf{z}_1 \mathbf{y}_1^* & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{z}_2 \mathbf{y}_2^* & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{z}_3 \mathbf{y}_3^* & \mathbf{0} & \dots & \mathbf{0} \\ \dots & \dots & \dots & \dots & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{z}_h \mathbf{y}_h^* \end{bmatrix}$$

It is readily checked that  $\mathbf{K}^{h-1} + \mathbf{K}^{h-2} + \dots + \mathbf{K} + \mathbf{I}$  is equal to

$$\begin{bmatrix} \mathbf{I} & \mathbf{K}_1 & \mathbf{K}_1 \mathbf{K}_2 & \mathbf{K}_1 \mathbf{K}_2 \mathbf{K}_3 & \dots & \mathbf{K}_1 \mathbf{K}_2 \dots \mathbf{K}_{h-1} \\ \mathbf{K}_2 \dots \mathbf{K}_{h-1} \mathbf{K}_h & \mathbf{I} & \mathbf{K}_2 & \mathbf{K}_2 \mathbf{K}_3 & \dots & \mathbf{K}_2 \dots \mathbf{K}_{h-1} \\ \mathbf{K}_3 \dots \mathbf{K}_{h-1} \mathbf{K}_h & \mathbf{K}_3 \dots \mathbf{K}_{h-1} \mathbf{K}_h \mathbf{K}_1 & \mathbf{I} & \mathbf{K}_3 & \dots & \mathbf{K}_3 \dots \mathbf{K}_{h-1} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \mathbf{K}_h & \mathbf{K}_h \mathbf{K}_1 & \mathbf{K}_h \mathbf{K}_1 \mathbf{K}_2 & \mathbf{K}_h \mathbf{K}_1 \mathbf{K}_2 \mathbf{K}_3 & \dots & \mathbf{I} \end{bmatrix}$$

Computing the product of these last two expressions and simplifying using the formulas for  $\mathbf{y}_i^*$ , we arrive at the desired formula for  $r = 0$ .

$$\begin{aligned} \lim_{q \rightarrow \infty} \frac{\rho^h - 1}{\rho^{hq} - 1} \sum_{i=0}^{hq-1} \mathbf{K}^i &= \begin{bmatrix} \mathbf{z}_1 \mathbf{y}_1^* & \rho \mathbf{z}_1 \mathbf{y}_2^* & \rho^2 \mathbf{z}_1 \mathbf{y}_3^* & \rho^3 \mathbf{z}_1 \mathbf{y}_4^* & \dots & \rho^{h-1} \mathbf{z}_1 \mathbf{y}_h^* \\ \rho^{h-1} \mathbf{z}_2 \mathbf{y}_1^* & \mathbf{z}_2 \mathbf{y}_2^* & \rho \mathbf{z}_2 \mathbf{y}_3^* & \rho^2 \mathbf{z}_2 \mathbf{y}_4^* & \dots & \rho^{h-2} \mathbf{z}_2 \mathbf{y}_h^* \\ \rho^{h-2} \mathbf{z}_3 \mathbf{y}_1^* & \rho^{h-1} \mathbf{z}_3 \mathbf{y}_2^* & \mathbf{z}_3 \mathbf{y}_3^* & \rho \mathbf{z}_3 \mathbf{y}_4^* & \dots & \rho^{h-3} \mathbf{z}_3 \mathbf{y}_h^* \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \rho \mathbf{z}_h \mathbf{y}_1^* & \rho^2 \mathbf{z}_h \mathbf{y}_2^* & \rho^3 \mathbf{z}_h \mathbf{y}_3^* & \rho^4 \mathbf{z}_h \mathbf{y}_4^* & \dots & \mathbf{z}_h \mathbf{y}_h^* \end{bmatrix} \\ &= \mathbf{Z} \Theta \mathbf{Y} \end{aligned}$$

It is readily verified that  $\mathbf{KZ} = \rho\mathbf{ZP}$ , so  $\mathbf{K}^r\mathbf{Z} = \rho^r\mathbf{ZP}^r$ . Thus we have for any  $r$ ,

$$\begin{aligned} \lim_{h \rightarrow \infty} \frac{\rho^h - 1}{\rho^{hq} - 1} \mathbf{K}^r \sum_{i=0}^{hq-1} \mathbf{K}^i &= \mathbf{K}^r \mathbf{Z} \Theta \mathbf{Y} \\ &= \rho^r \mathbf{ZP}^r \Theta \mathbf{Y} \end{aligned}$$

□

For the following theorem, let  $\mathbf{\Lambda}_i$  be the submatrix of  $\mathbf{\Lambda}$  with rows indexed by elements of  $E_i$  and let  $\mathbf{c}_i = \mathbf{y}_i^* \mathbf{\Lambda}_i$ , so that

$$\begin{bmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \\ \mathbf{c}_3 \\ \dots \\ \mathbf{c}_h \end{bmatrix} = \begin{bmatrix} \mathbf{y}_1^* & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{y}_2^* & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{y}_3^* & \dots & \mathbf{0} \\ \dots & \dots & \dots & \dots & \dots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{y}_h^* \end{bmatrix} \begin{bmatrix} \mathbf{\Lambda}_1 \\ \mathbf{\Lambda}_2 \\ \mathbf{\Lambda}_3 \\ \dots \\ \mathbf{\Lambda}_h \end{bmatrix}$$

For notational convenience, we define  $\mathbf{c}_i$  for any integer by reducing the index modulo  $h$  using representatives  $1, 2, \dots, h$ .

**Theorem 4.4.** *For  $r \in \{0, \dots, h-1\}$  and  $e \in E_i$ ,*

$$\lim_{q \rightarrow \infty} \left( x_e^{(hq+r)} \right)^{\frac{1}{\rho^r} \cdot \frac{\rho^h - 1}{\rho^{hq} - 1}} = \left( \prod_{j=0}^{h-1} (\mathbf{u}^{\mathbf{c}_{r+i+j}})^{\rho^j} \right)^{\mathbf{z}_e}$$

*The sum-product algorithm converges if and only if the following products have the same parity as  $r$  varies.*

$$U_r = \prod_{j=0}^{h-1} (\mathbf{u}^{\mathbf{c}_{r+j}})^{\rho^j}$$

*Proof.* Dividing the equation of Lemma 4.3 by  $\rho^r$  and multiplying by  $\mathbf{\Lambda}$  we have  $\frac{1}{\rho^r} \cdot \frac{\rho^h - 1}{\rho^{hq} - 1} \mathbf{A}^{(hq+r)} \approx \mathbf{ZP}^r \Theta \mathbf{Y} \mathbf{\Lambda}$ . The matrix  $\mathbf{Y} \mathbf{\Lambda}$  is  $h \times |L|$  with  $i^{\text{th}}$  row  $\mathbf{c}_i$ , so the  $i^{\text{th}}$  row of  $\Theta \mathbf{Y} \mathbf{\Lambda}$  is  $\mathbf{c}_i + \mathbf{c}_{i+1}\rho + \mathbf{c}_{i+2}\rho^2 + \dots + \mathbf{c}_{i+h-1}\rho^{h-1}$  where subscripts are computed modulo  $h$  using representatives  $1, \dots, h$ . The  $i^{\text{th}}$  row of  $\mathbf{P}^r \Theta \mathbf{Y} \mathbf{\Lambda}$  is  $\mathbf{c}_{r+i} + \mathbf{c}_{r+i+1}\rho + \mathbf{c}_{r+i+2}\rho^2 + \dots + \mathbf{c}_{r+i+h-1}\rho^{h-1}$ . Thus if  $e \in E_i$  then

$$\lim_{q \rightarrow \infty} \frac{1}{\rho^r} \frac{\rho^h - 1}{\rho^{hq} - 1} \mathbf{a}_e^{(hq+r)} = \mathbf{z}_e \sum_{j=0}^{h-1} \mathbf{c}_{r+i+j} \rho^j$$

and

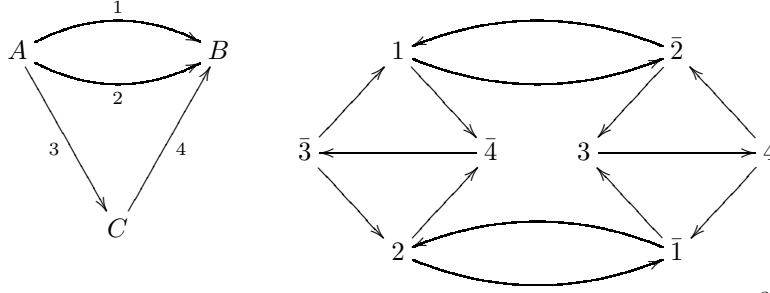
$$\lim_{q \rightarrow \infty} \left( x_e^{(hq+r)} \right)^{\frac{1}{\rho^r} \cdot \frac{\rho^h - 1}{\rho^{hq} - 1}} = \left( \prod_{j=0}^{h-1} (\mathbf{u}^{\mathbf{c}_{r+i+j}})^{\rho^j} \right)^{\mathbf{z}_e}$$

For convergence of  $x_e$ , the value of  $\mathbf{z}_e$  is irrelevant. For  $i = 0$  we get the condition stated in the Theorem, but for other  $i$  we get the same set of products since the subscripts on  $\mathbf{c}_i$  are computed modulo  $h$ . □

## 5. EXAMPLES

**Example 5.1.** Let  $G$  be  $d$ -regular with  $n$  vertices and therefore  $nd$  edges. Then  $\tilde{G}$  and its adjacency matrix  $\mathbf{K}$  are  $(d-1)$ -regular. Let us assume that  $\mathbf{K}$  is primitive. Then the Perron eigenvalue of  $\mathbf{K}$  is  $\rho = d-1$  and the Perron vectors, of length  $nd$ , are constant, that is, multiples of  $[1, 1, \dots, 1]$  or its transpose. We may take  $\mathbf{z}$  to be 1 in each component and  $\mathbf{y}^*$  to be  $1/nd$  in each component. The matrix  $\mathbf{\Lambda}$  is  $nd \times n$  and has  $d$  ones in each column, so  $\mathbf{c} = \mathbf{y}^* \mathbf{\Lambda} = [1/n, 1/n, \dots, 1/n]$ . Theorem 4.1 says that the SPA converges to 0 when  $\left(\prod_{\ell \in L} u_\ell\right)^{1/n} < 1$ , or, equivalently, when  $\prod_{\ell \in L} u_\ell < 1$ . The SPA converges to  $\infty$  when  $\prod_{\ell \in L} u_\ell > 1$ . The convergence is roughly exponential with base  $\prod_{\ell \in L} u_\ell > 1$  and exponent  $\rho/n$ .

**Example 5.2.** Consider the graph  $G$  and its flow graph  $\tilde{G}$  below. There are 3 vertices,  $A, B, C$  and 4 conjugate pairs of edges in  $G$ ; one edge of each pair is shown.



The matrix  $\mathbf{K}$  is  $8 \times 8$  and its largest eigenvalue,  $\rho$ , is the positive root of  $x^3 - x^2 - 2$ .

Taking the edges in the order 1, 2, 3, 4,  $\bar{1}$ ,  $\bar{2}$ ,  $\bar{3}$ ,  $\bar{4}$ , the eigenvector  $\mathbf{y}^*$  is, up to multiple,  $[1, 1, \rho-1, \rho^2-\rho, 1, 1, \rho^2-\rho, \rho-1]$ . Taking the vertices in alphabetical order, the transpose of  $\mathbf{\Lambda}$  is

$$\mathbf{\Lambda}^T = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

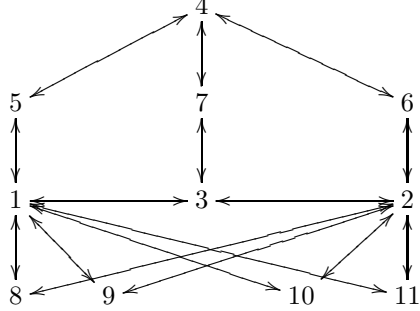
so  $\mathbf{c} = \mathbf{y}^* \mathbf{\Lambda} = [\rho+1, \rho+1, 2(\rho^2-\rho)]$ . Since  $\rho \approx 1.6956$ , we have  $\mathbf{y}^* \approx [1, 1, 0.7, 1.2, 1, 1, 1.2, 0.7]$  and  $\mathbf{c} \approx [2.7, 2.7, 2.4]$ . Not surprisingly, based on the higher degree of the nodes  $A$  and  $B$ , the input values for  $A$  and  $B$  have a greater influence on convergence than does  $C$ . Notice that it is possible for the SPA to return the codeword 000 when 111 is most likely. Such is the case for  $u_A = u_B = 0.5$ , and  $u_C = 4.5$ .

The last example might lead one to suppose that higher-degree nodes have a greater influence on convergence than lower-degree nodes—that is, if  $\ell$  has higher degree than  $\ell'$ , then  $\mathbf{c}_\ell > \mathbf{c}_{\ell'}$ —but this is not necessarily the case.

**Example 5.3.** Consider the graph below. Up to scaling,

$$\mathbf{c} \approx [2.1, 2.1, 1.7, 1.2, 1.8, 1.3, 1.3, 1.1, 1.6, 1.6, 1.6]$$

Although the node at the top has higher degree than the nodes at the bottom, the corresponding component in  $\mathbf{c}$  is smaller; *e.g.*,  $\mathbf{c}_4 < \mathbf{c}_8$ .



**Example 5.4.** Let  $G$  be bipartite, so that  $L$  is the disjoint union of  $L_1$  and  $L_2$  and  $E$  is the disjoint union of  $E_1$  and  $E_2$  with all edges in  $E_i$  having source in  $L_i$ . Conjugation gives a bijection of  $E_1$  with  $E_2$ . Enumerating the edges of  $E_1$  before those of  $E_2$  and the vertices of  $L_1$  before those in  $L_2$  we have

$$\mathbf{K} = \begin{bmatrix} 0 & \mathbf{K}_1 \\ \mathbf{K}_2 & 0 \end{bmatrix} \quad \mathbf{\Lambda} = \begin{bmatrix} \mathbf{\Lambda}_1 & 0 \\ 0 & \mathbf{\Lambda}_2 \end{bmatrix}$$

Notice that  $\mathbf{\Lambda}_i$  differs slightly from the notation used in the theorem, where  $\mathbf{\Lambda}_i$  is the upper  $|E_1|$  rows of  $\mathbf{\Lambda}$ .

Let us assume that the index of imprimitivity of  $\mathbf{K}$  is  $h = 2$ . Suppose that  $G$  is  $(d_1, d_2)$ -regular and that  $n_i = |L_i|$ , so that  $|E_i| = n_i d_i$ , and  $n_1 d_1 = n_2 d_2$ . Each edge in  $E_1$  has  $d_1 - 1$  non-conjugate edges which feed into it, so  $\mathbf{K}_1$  is  $(d_1 - 1)$ -regular. Similarly,  $\mathbf{K}_2$  is  $(d_2 - 1)$ -regular. Each row and each column of the matrices  $\mathbf{K}_1 \mathbf{K}_2$  and  $\mathbf{K}_2 \mathbf{K}_1$  sums to  $(d_1 - 1)(d_2 - 1)$ , so  $\mathbf{K}_1 \mathbf{K}_2$  and  $\mathbf{K}_2 \mathbf{K}_1$  have Perron eigenvalue  $(d_1 - 1)(d_2 - 1)$  and Perron eigenvectors that are constant. The Perron eigenvalue of  $\mathbf{K}$  is  $\rho = \sqrt{(d_1 - 1)(d_2 - 1)}$ . We take  $\mathbf{y}_1^*$  to be  $1/\sqrt{d_1 - 1}$  in each component. Then  $\mathbf{y}_2^* = \rho^{-1} \mathbf{y}_1^* \mathbf{K}_1$  is  $1/\sqrt{d_2 - 1}$  in each component.

Since  $\mathbf{\Lambda}_i$  has  $d_i$  1's in each column,  $\mathbf{y}_i^* \mathbf{\Lambda}_i$  is  $d_i/\sqrt{d_i - 1}$  in each component. In the notation of Theorem 4.4 we have

$$(\mathbf{c}_i)_\ell = \begin{cases} \frac{d_i}{\sqrt{d_i - 1}} & \text{if } \ell \in E_i \\ 0 & \text{otherwise} \end{cases}$$

$$U_0 = \mathbf{u}^{\mathbf{c}_2} (\mathbf{u}^{\mathbf{c}_1})^\rho = \left( \prod_{\ell \in L_2} u_\ell \right)^{\frac{d_2}{\sqrt{d_2 - 1}}} \left( \prod_{\ell \in L_1} u_\ell \right)^{d_1 \sqrt{d_2 - 1}}$$

$$U_1 = \mathbf{u}^{\mathbf{c}_1} (\mathbf{u}^{\mathbf{c}_2})^\rho = \left( \prod_{\ell \in L_1} u_\ell \right)^{\frac{d_1}{\sqrt{d_1 - 1}}} \left( \prod_{\ell \in L_2} u_\ell \right)^{d_2 \sqrt{d_1 - 1}}$$

Notice that for  $U_0$  the ratio of the exponents appearing in the formula is  $d_1(d_2 - 1)/d_2$ , while for  $U_1$  it is  $d_2(d_1 - 1)/d_2$ . The SPA converges if and only if  $U_0$  and  $U_1$  have the same parity.

If  $G$  is regular,  $d_1 = d_2 = d$ , then  $n_1 = n_2$  and  $\rho = d - 1$ . The SPA converges if and only if  $\left( \prod_{\ell \in L_1} u_\ell \right) \left( \prod_{\ell \in L_2} u_\ell \right)^{d-1}$  and  $\left( \prod_{\ell \in L_1} u_\ell \right)^{d-1} \left( \prod_{\ell \in L_2} u_\ell \right)$  have the same parity.

It is possible to have higher index of primitivity; Figure 5 gives several examples.



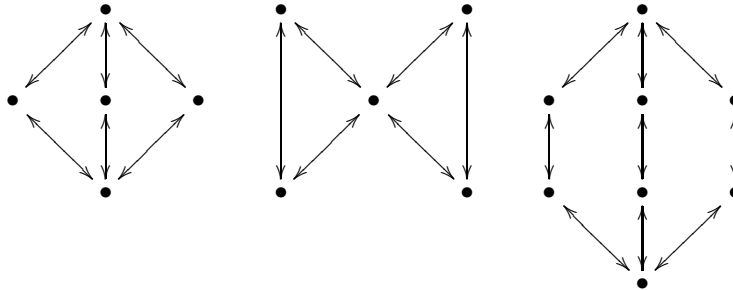


FIGURE 4. Three graphs  $G$  such that the index of imprimitivity  $h$  of the adjacency matrix of  $\tilde{G}$  is larger than 2. From left to right,  $h = 4$ ,  $h = 3$ ,  $h = 6$ .

## 6. TRAPPING SETS

This section extends results of the previous sections to trapping sets. An  $(a, b)$  *trapping set* is a subgraph of a bipartite graph consisting of  $a$  bit nodes and all their neighboring check nodes such that  $b$  check nodes on the subgraph have odd degree. Richardson introduced the term “trapping set” in [17], where he used a combination of simulation and combinatorial analysis of trapping sets to predict error floors. MacKay and Postol used the term *near-codeword* for the same phenomena in [12], since for small  $b$ , a vector with support on the  $a$  bit nodes is “nearly” a codeword. McKay and Postol attributed decoding failure of a Margulis code in the error floor region to near-codewords. A great deal of research has been focused on trapping sets. An edge swapping construction that eliminates small trapping sets was shown to lower the error floor in [7, 4]. Analysis of trapping sets led to accurate prediction of decoding performance of LDPC codes on the binary symmetric channel using the Gallager A algorithm in [2]. Counts of small trapping sets were shown to be good predictors for performance of the SPA in [10]. Planjery et al [15, 16] developed a message-passing algorithm for the binary symmetric channel that is similar to belief propagation, but is designed to overcome errors due to trapping sets.

We are going to identify the conditions under which the SPA converges on a trapping set, but first we must properly frame the problem. As Richardson and others have noted (see for example the “trapping set ontology” of [22]) simulation has shown that the trapping sets that affect the error floor invariably have check nodes of degree at most 2. The graph obtained by removing the degree-1 checks from such a trapping set is either a cycle or one of the graphs treated in Section 4; we will call this graph the *core* of the trapping set. Note that considering the SPA on a detached trapping set yields nothing useful: In the SPA, a check of degree 1 causes all messages from the neighboring bit to other checks to be 0 after the first iteration of the algorithm, and this causes the SPA to eventually converge to the zero codeword on the trapping set. To avoid this, we can modify the trapping set by adding a bit node to each check node of degree 1. This “virtual bit” serves as the communication link between the ambient graph and the core graph. Starting with an  $(a, b)$  trapping set we now have a bipartite graph with all check nodes of degree 2 and  $b$  bit nodes of degree 1. It will ease our analysis to add a new degree-1

bit node *for each bit* of our core graph, along with a check node connecting the two. This might seem to violate the essential interest in trapping sets, the small number  $b$  of odd degree check nodes, but, as we now show, the behavior of the SPA on the trapping set may be easily deduced from our somewhat larger graph.

Consider for a moment the SPA on an arbitrary bipartite graph  $B'$  having a bit node  $\ell'$  of degree 1. It is straightforward to verify that if  $u'_{\ell'} = 1$ , the SPA on  $B'$  has the same edge messages at any iteration  $t$ —as a function of the  $u_{\ell}$  for  $\ell \in L$ —as the SPA on the graph  $B$  obtained from  $B'$  by removing  $\ell'$  and its incident edge. In other words, setting  $u_{\ell'} = 1$ , for  $\ell'$  a leaf node, effectively removes  $\ell'$  from the algorithm. Seen another way, suppose we start with a bipartite graph  $B$  and add a bit node  $\ell'$  and an edge to create a new graph  $B'$ . The behavior of the SPA on  $B$  can be recovered from the behavior of the SPA on  $B'$  by setting  $u_{\ell'} = 1$ .

Our strategy to understand the behavior of the SPA on trapping sets is therefore the following: we start with a bipartite graph  $B = (E, L, R, \lambda, \rho)$  satisfying the properties of Section 4, and the associated undirected graph  $G$ . We let  $\mathbf{A}$  and  $\mathbf{K}$  be the edge-vertex incidence matrix and flow matrix, for  $G$ . Let  $L = \{\ell_1, \dots, \ell_n\}$ ; now, create a new set of bit nodes  $L' = \{\ell'_1, \dots, \ell'_m\}$  along with new edges  $E_1 = \{e_1, \dots, e_m\}$  and  $E_2 = \{\bar{e}_1, \dots, \bar{e}_m\}$  so that edge  $e_i$  goes from  $\ell'_i$  to  $\ell_i$ , and  $\bar{e}_i$  is its conjugate edge going from  $\ell_i$  to  $\ell'_i$ . Thus, we have effectively attached a leaf node to each node of  $G$  to create a new graph; call this graph  $G'$  and the associated bipartite graph  $B'$ . Since all check nodes have degree 2, the SPA passes monomials in the input values  $u_{\ell}$ , so may analyze the SPA on the graph we have constructed via the local sum algorithm on the associated undirected graph. The main result is the following.

**Theorem 6.1.** *With the preceding notation, the sum-product algorithm on  $B'$  with inputs  $u_i$  for  $\ell_i \in L$  and  $u'_i$  for  $\ell'_i \in L'$  converges if and only if the sum-product algorithm on  $B$  with input  $u'_i u_i^{\rho}$  converges. Here  $\rho$  is the Perron eigenvalue for the flow matrix of  $B$ .*

Our primary interest in this theorem is the application to trapping sets, in which case we set  $u'_i = 1$  for any  $\ell$  which is not connected to a check node of degree 1 in the trapping set.

*Proof.* The edge-vertex matrix for our new graph is

$$\mathbf{A}' = \begin{bmatrix} \mathbf{I} & 0 \\ 0 & \mathbf{A} \\ 0 & \mathbf{I} \end{bmatrix}$$

The flow matrix  $\mathbf{K}'$  is a  $3 \times 3$  block matrix, corresponding to the partition of the edges into  $E_1$ ,  $E$  and  $E_2$ . The rows for  $e_i \in E_1$  are 0, because  $\lambda(e_i)$  is a leaf, so no edge terminates at  $\lambda(e_i)$  except  $\bar{e}_i$ . Similarly, columns for  $\bar{e}_i \in E_2$  are 0, since the edges in  $E_2$  terminate in a leaf. The block matrix for  $E_2 \times E_1$  is 0, since we don't allow flow into conjugate edges. Since  $e_i \in E_1$  terminates in  $\ell_i$ ,  $e_i$  flows into the edges of  $E$  that have source  $\ell_i$ . Thus the block matrix for  $E \times E_1$  is exactly the same as the edge-vertex incidence matrix for  $G$ , namely,  $\mathbf{A}$ . The matrix for  $E_2 \times E$  is similar, the edges that flow into  $\bar{e}_i$ , are the edges in  $E$  that terminate at  $\ell_i$ . Let  $\mathbf{T}$  be the  $|E| \times |E|$  matrix for defined by  $\mathbf{T}_{e,\bar{e}} = 1$  and  $T$  is 0 elsewhere, so  $\mathbf{T}^2 = \mathbf{I}$ . The  $E_2 \times E$  matrix is  $\mathbf{A}^{\dagger} \mathbf{T}$ . Finally, the  $E \times E$  portion of  $\mathbf{K}'$  is  $\mathbf{K}$  the flow matrix for  $G$ . Thus we have

$$\mathbf{K}' = \begin{bmatrix} 0 & 0 & 0 \\ \mathbf{\Lambda} & \mathbf{K} & 0 \\ 0 & \mathbf{\Lambda}^\dagger \mathbf{T} & 0 \end{bmatrix}$$

The matrix  $\mathbf{T}$  defined gives a useful expression for the flow matrix of the core graph, and for its Perron eigenvector  $\mathbf{y}^*$ . It is readily verified that  $\mathbf{K} = \mathbf{\Lambda} \mathbf{\Lambda}^\dagger \mathbf{T} - \mathbf{T}$ . Then  $\mathbf{z}^\dagger \mathbf{T}$  is a left eigenvector for  $\mathbf{K}$ :

$$\begin{aligned} (\mathbf{\Lambda} \mathbf{\Lambda}^\dagger \mathbf{T} - \mathbf{T}) \mathbf{z} &= \rho \mathbf{z}, \quad \text{so} \\ \mathbf{z}^\dagger (\mathbf{T} \mathbf{\Lambda} \mathbf{\Lambda}^\dagger - \mathbf{T}) &= \rho \mathbf{z}^\dagger \end{aligned}$$

Multiplying on the right by  $\mathbf{T}$  and noting that  $\mathbf{T}^2 = \mathbf{I}$ ,

$$\mathbf{z}^\dagger \mathbf{T} (\mathbf{\Lambda} \mathbf{\Lambda}^\dagger \mathbf{T} - \mathbf{T}) = \rho \mathbf{z}^\dagger \mathbf{T}$$

Thus we may take  $\mathbf{y}^* = \mathbf{z}^\dagger \mathbf{T}$  provided we normalize  $\mathbf{z}$  so that  $\mathbf{z}^\dagger \mathbf{T} \mathbf{z} = 1$ .

Consider the case when  $\mathbf{K}$  is primitive (the imprimitive case is similar). Using the approximation of Section 4, for  $t > 1$ ,

$$\begin{aligned} (\mathbf{K}')^t &= \begin{bmatrix} 0 & 0 & 0 \\ \mathbf{K}^{t-1} \mathbf{\Lambda} & \mathbf{K}^t & 0 \\ \mathbf{\Lambda}^\dagger \mathbf{T} \mathbf{K}^{t-2} \mathbf{\Lambda} & \mathbf{\Lambda}^\dagger \mathbf{T} \mathbf{K}^{t-1} & 0 \end{bmatrix} \\ &\approx \begin{bmatrix} 0 & 0 & 0 \\ \rho^{t-1} \mathbf{z} \mathbf{y}^* \mathbf{\Lambda} & \rho^t \mathbf{z} \mathbf{y}^* & 0 \\ \rho^{t-2} \mathbf{\Lambda}^\dagger \mathbf{T} \mathbf{z} \mathbf{y}^* \mathbf{\Lambda} & \rho^{t-1} \mathbf{\Lambda}^\dagger \mathbf{T} \mathbf{z} \mathbf{y}^* & 0 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 0 & 0 \\ \rho^{t-1} \mathbf{z} \mathbf{z}^\dagger \mathbf{T} \mathbf{\Lambda} & \rho^t \mathbf{z} \mathbf{z}^\dagger \mathbf{T} & 0 \\ \rho^{t-2} \mathbf{\Lambda}^\dagger \mathbf{T} \mathbf{z} \mathbf{z}^\dagger \mathbf{T} \mathbf{\Lambda} & \rho^{t-1} \mathbf{\Lambda}^\dagger \mathbf{T} \mathbf{z} \mathbf{z}^\dagger \mathbf{T} & 0 \end{bmatrix} \end{aligned}$$

Now compute

$$\begin{aligned} (\mathbf{A}')^{(t)} &= \left( \sum_{i=0}^{t-1} (\mathbf{K}')^i \right) \mathbf{\Lambda}' \\ &\approx \begin{bmatrix} \mathbf{I} & 0 \\ \frac{\rho^{t-1}-1}{\rho-1} \mathbf{z} \mathbf{z}^\dagger \mathbf{T} \mathbf{\Lambda} & \frac{\rho^t-1}{\rho-1} \mathbf{z} \mathbf{z}^\dagger \mathbf{T} \mathbf{\Lambda} \\ \frac{\rho^{t-2}-1}{\rho-1} \mathbf{\Lambda}^\dagger \mathbf{T} \mathbf{z} \mathbf{z}^\dagger \mathbf{T} \mathbf{\Lambda} & \frac{\rho^{t-1}-1}{\rho-1} \mathbf{\Lambda}^\dagger \mathbf{T} \mathbf{z} \mathbf{z}^\dagger \mathbf{T} \mathbf{\Lambda} \end{bmatrix} \end{aligned}$$

Let  $\mathbf{c} = \mathbf{y}^* \mathbf{\Lambda} = \mathbf{z}^\dagger \mathbf{T} \mathbf{\Lambda}$  (a  $1 \times n$  vector, since  $n = |L|$ ). This is the same vector we used for the core graph. Taking the limit,

$$\lim_{t \rightarrow \infty} \frac{\rho-1}{\rho^{t-1}-1} \mathbf{A}^{(t)} = \begin{bmatrix} 0 & 0 \\ \mathbf{z} \mathbf{c} & \rho \mathbf{z} \mathbf{c} \\ \rho^{-1} \mathbf{c}^\dagger \mathbf{c} & \mathbf{c}^\dagger \mathbf{c} & 0 \end{bmatrix}$$

Now let  $\mathbf{u}$  be the vector of variables for  $L$  and  $\mathbf{u}'$  the vector of variables for  $L'$ . We have for  $e \in E$

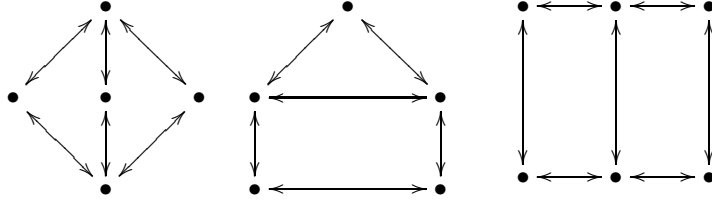
$$\begin{aligned} \lim_{t \rightarrow \infty} \frac{\rho-1}{\rho^t-1} \mathbf{a}_e^{(t)} &= [\mathbf{z} \mathbf{c} \quad \rho \mathbf{z} \mathbf{c}] \\ \lim_{t \rightarrow \infty} \left( x_e^{(t)} \right)^{\frac{\rho-1}{\rho^t-1}} &= \lim_{t \rightarrow \infty} \left( (\mathbf{u}')^{\mathbf{c}} \mathbf{u}^{\rho \mathbf{c}} \right)^{\mathbf{z}_e} \end{aligned}$$

For  $\bar{e}_i \in E_2$  recall that  $\bar{e}_i$  is the edge from  $\ell_i \in L$  to  $\ell'_i \in L'$

$$\lim_{t \rightarrow \infty} \left( x_{\bar{e}_i}^{(t)} \right)^{\frac{\rho-1}{\rho^t-1}} = \lim_{t \rightarrow \infty} \left( (\mathbf{u}')^{\rho^{-1}} \mathbf{c} \mathbf{u}^{\mathbf{c}} \right)^{c_i}$$

For any edge  $e$ ,  $x_e$  goes to 0 if  $(\mathbf{u}')^{\mathbf{c}} \mathbf{u}^{\rho \mathbf{c}} < 1$  and to  $\infty$  if  $(\mathbf{u}')^{\mathbf{c}} \mathbf{u}^{\rho \mathbf{c}} > 1$ . Noting that  $(\mathbf{u}')^{\mathbf{c}} \mathbf{u}^{\rho \mathbf{c}} = \prod_{\ell \in L} (u'_\ell (u_\ell)^\rho)^{c_\ell}$  establishes the theorem.  $\square$

**Example 6.2.** Consider each of the graphs below as the core of a trapping set in which all bit nodes have degree 3. Consider the bit nodes of degree 2 in the figure as attached to another bit node, which is not shown. All three appear in the trapping set ontology [22], and the first graph is a prominent example in the papers of Planjery, Vasic and co-authors.



The first two graphs lead to  $(5, 3)$  trapping sets and the third graph leads to a  $(6, 2)$  trapping set. The flow matrix for the  $(5, 3)$  with girth 8 has index of imprimitivity 4 with  $\rho = \sqrt{2}$ . The flow matrix for the  $(5, 3)$  with girth 6 is primitive, since the gcd of admissible cycles is 1, and  $\rho = 1.424$ , slightly larger than  $\sqrt{2}$ . The flow matrix for the  $(6, 2)$  graph has index of imprimitivity 2, with  $\rho = 1.353$ .

The first graph is particularly interesting. Label the edges from the top node down as 1, 2, 3 and from the bottom node up as 4, 5, 6, in each case starting from the left to the right. Order the edges as follows: 1, 2, 3,  $\bar{1}$ ,  $\bar{2}$ ,  $\bar{3}$ , 4, 5, 6,  $\bar{4}$ ,  $\bar{5}$ ,  $\bar{6}$ . Let  $\mathbf{I}$  be a  $3 \times 3$  identity matrix, let  $\mathbf{1}$  be a  $3 \times 1$  vector of with one in each entry, and let  $\mathbf{J} = \mathbf{1}\mathbf{1}^\dagger$ . Let  $\mathbf{0}$  be a zero matrix or zero vector as determined by context. With an appropriate ordering for  $L$ , we get the following structural matrices.

$$\mathbf{A} = \begin{bmatrix} \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \\ \mathbf{I} & \mathbf{0} & \mathbf{0} \end{bmatrix} \quad \mathbf{T} = \begin{bmatrix} \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} \end{bmatrix}$$

$$\mathbf{A}^\dagger \mathbf{T} = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{1}^\dagger & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1}^\dagger \end{bmatrix} \quad \mathbf{K} = \begin{bmatrix} \mathbf{0} & \mathbf{J} - \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{J} - \mathbf{I} \\ \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}$$

One might expect that the  $(5, 3)$  trapping set with girth 8 performs better under the sum-product algorithm than the  $(5, 3)$  trapping set with girth 6, but the reverse is true, as can be seen from the performance curves in Figure 5. The primitivity of the flow matrix of the girth 6 graph makes it less susceptible to channel noise.

We conclude this section with a suggested program that may result in closed-form probability calculations for the decoding failure on any graph, analogous to the results of Chilappagari et al [2] on the binary symmetric channel (see also [15, 16]). A key to their analysis of decoding algorithms is an “isolation assumption,” which gives conditions under which the behavior of the decoding algorithm on the trapping

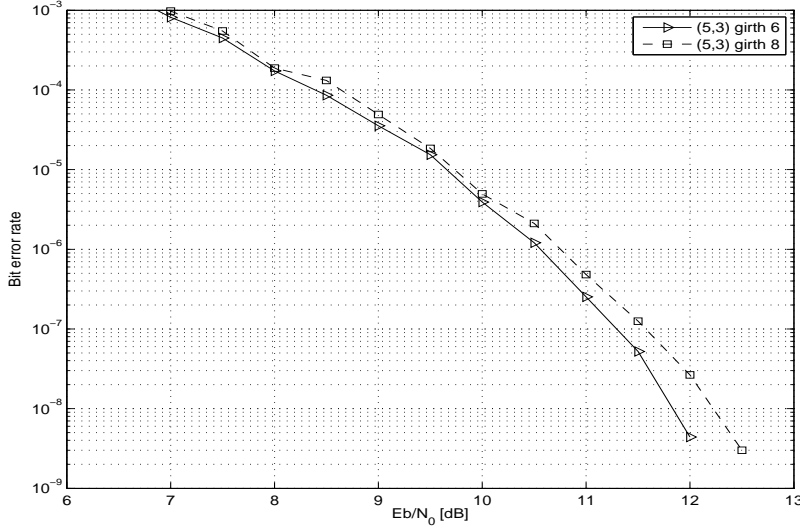


FIGURE 5. Comparison of the performance of the SPA on the core graph of two  $(5,3)$  trapping sets. The termination criterion was  $\epsilon = 10^{-8}$ .

set can be isolated from the behavior on the rest of the graph. The essence of the isolation assumption is that for some length  $M$  there is no path through the complement of the trapping set, between nodes of the trapping set. Thus for  $M$  iterations of the algorithm, the affect of one edge message of the trapping set on another edge message of the trapping set is completely determined by the message passing of the trapping set and unaffected by paths in the complement.

In order to simplify the discussion and make definite statements, we will assume that all bits in the original graph have degree 3. Each bit in the core of a trapping set has degree 2 or 3, so it is “missing” at most one check. When we add in the new bits  $\ell'$  to the core, the core bits now have degree 3 or 4. For those of degree 4, the check we added does not correspond to a check in actual graph, so set the  $u_{\ell'}$  to 1; the remaining  $\ell'$  – actually, their attendant checks – can be thought of as representing the incoming information to the trapping set from the rest of the graph. Under the assumption, then, that these incoming messages are essentially independent of the outgoing messages passed out of the trapping set, one should be able to obtain an expression for the probability that some bit is in error after the  $t^{\text{th}}$  iteration of the algorithm via the initial distribution on the core bits, density-evolution analysis on the check-to-bit messages from the new checks for which  $u_{\ell'}$  was not set equal to 1 (either via the methods of [19] or via Monte Carlo simulation), and the formula in Theorem 6.1.

## 7. CONCLUDING REMARKS

In this section we summarize our results on regions of convergence, we comment on covering graphs, and we present some experimental results illustrating the preceding theory.

*Region of Convergence.* When the matrix  $\mathbf{K}$  is primitive, Theorem 4.1 shows that the SPA will converge except on a region of measure 0, defined by the equation  $\mathbf{u}^{\mathbf{c}} = 1$ . For regular graphs, the SPA is therefore a maximum likelihood decoder, since this hypersurface is  $\prod_{\ell \in L} u_{\ell} = 1$ , which corresponds to  $\prod_{\ell \in L} p_{\ell}(1) = \prod_{\ell \in L} p_{\ell}(0)$ . For an irregular graph, the components of  $\mathbf{c}$  differ, and the SPA may place more emphasis on certain  $u_{\ell}$ .

When the matrix  $\mathbf{K}$  is imprimitive, there is a region of positive measure on which the SPA does not converge. The simplest example is  $K_{3,2}$ , the complete bipartite graph on 3 bits and 2 checks. The SPA will converge to the all-0 codeword on the region defined by  $u_1^2 u_2 < 1$  and  $u_1 u_2^2 < 1$  and to the all-1 codeword on the region  $u_1^2 u_2 > 1$  and  $u_1 u_2^2 > 1$ . The SPA will not converge on the region between the curves  $u_1 u_2^2 = 1$  and  $u_1^2 u_2 = 1$ .

The secondary eigenvalues may also affect performance. Suppose we use the criterion for termination of the SPA defined earlier: decode to the 0-codeword if for all  $\ell \in L$ ,  $\hat{u}_{\ell} < \epsilon$  and to the all-1-codeword if for all  $\ell \in L$ ,  $\hat{u}_{\ell} > 1/\epsilon$ . If the secondary eigenvalues are close to the Perron eigenvalue or if there are many large secondary eigenvalues, it may take a longer time for the Perron vector to become dominant. Since  $\hat{u}_{\ell}$  involves large powers of the  $u_{\ell}$  (as  $t$  gets large) it tends toward extremes (0 or  $\infty$ ), and this may cause the algorithm to terminate early and incorrectly.

*Covering Graphs.* It is common to construct LDPC codes by using covering graphs, but these observations suggest that there may be an inherent weakness. One can show that a covering graph will inherit all the eigenvalues of a base graph. Thus, imprimitivity or a small spectral gap in a base graph yield the same properties in a covering graph. This argument reverses the usual concern with graph covers: that pseudo-codewords—essentially codewords from a covering graph of  $B$ —can influence the performance of the SPA on  $B$ . When check nodes have degree 2, the only pseudo-codewords are constant vectors, and the only extremal pseudo-codewords are the all-0 vector and the all-1 vector. In this case, analysis of pseudo-codewords can say nothing about variation in decoding performance. Yet there are clear differences due to imprimitivity, and to other, more subtle, properties, as the following examples illustrate.

*Some Experiments.* The bipartite graph derived from the following graph is a 2-fold cover of  $K_{3,2}$ .

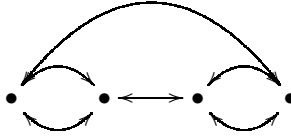


Figure 1 shows the bipartite graph derived from  $K_4$ , which is not a cover of  $K_{3,2}$ . The flow matrix for the 2-fold cover is imprimitive of index 2, while the flow matrix for the graph derived from  $K_4$  is primitive. Figure 6 shows performance of the SPA with convergence parameters  $\epsilon = 10^{-2}$ ,  $10^{-4}$  and  $10^{-8}$ . The curve for the bipartite graph from  $K_4$  maintains roughly a 0.5 dB gain over the 2-fold cover at all values of  $\epsilon$ . Figure 7 shows performance curves, using the same values for  $\epsilon$ , for a five-fold cover of  $K_{3,2}$ , which is imprimitive of index 2, and the bipartite graph derived from the Peterson graph, which has a primitive flow matrix. We see an improvement of roughly 0.5 dB at  $\epsilon = 10^{-8}$ , but with less discriminating choices of convergence parameter the Peterson graph actually performs worse.

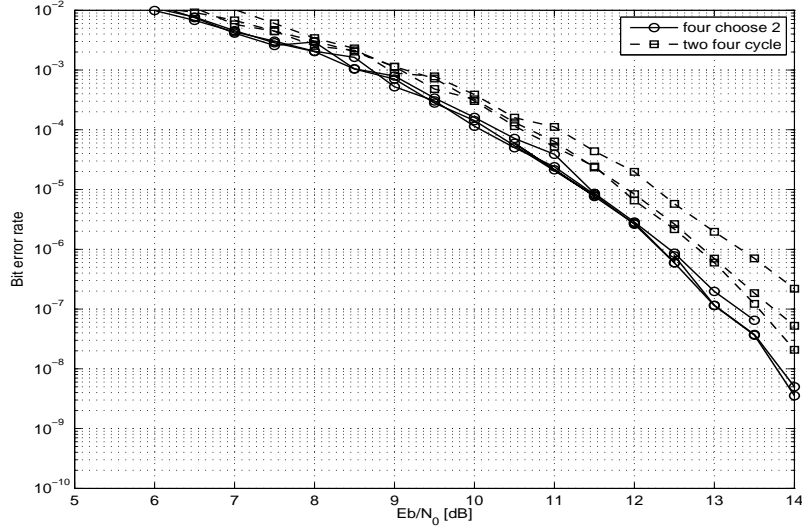


FIGURE 6. The performance of a 2-cover of  $K_{32}$  and the bipartite graph derived from  $K_4$  for  $\epsilon = 10^{-2}, 10^{-4}, 10^{-8}$ .

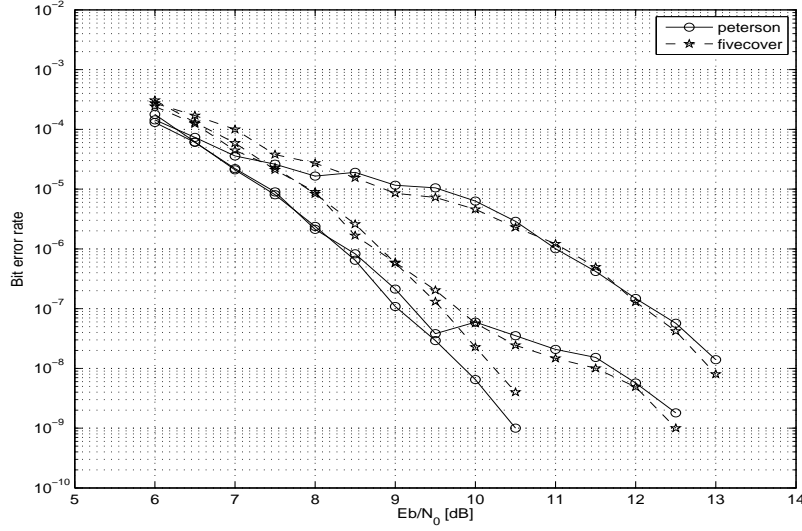
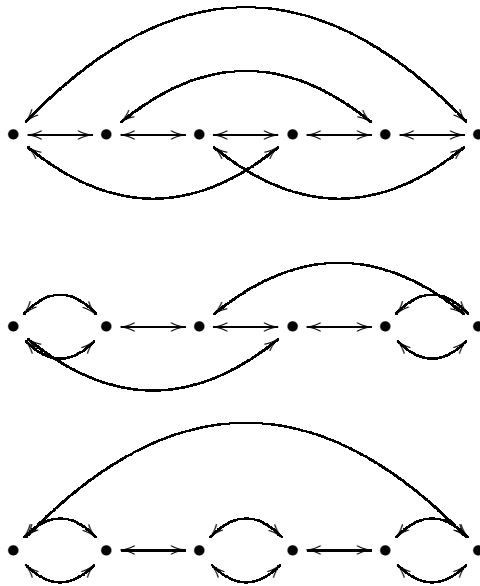


FIGURE 7. The performance of a 5-cover of  $K_{32}$  and the bipartite graph derived from the Peterson graph for  $\epsilon = 10^{-2}, 10^{-4}, 10^{-8}$ .

Both  $K_4$  and the Peterson graph are 3-*cages*, that is, 3-regular graphs having the minimum number of nodes for their respective girths, which are 3 and 5. (Note that the bipartite graphs derived from these graphs have girth that is twice as large.) One might attribute the performance gain to the large girth, but by itself this is not an assurance of asymptotic optimality. Below are the connected three-to-one

covers of  $K_{3,2}$ . One has girth 4; one has two 2-cycles; and one has three 2-cycles. (Again, double these values for the bipartite graph derived from these graphs.)



The girth-4 graph is actually a  $(3, 4)$ -cage. Since all these graphs are covers of  $K_{3,2}$  they are all imprimitive and have the same asymptotic performance, as can be seen in Figure 8, where the performance curves overlay each other for  $\epsilon = 10^{-8}$ . Observe that for  $\epsilon = 10^{-2}$  and  $10^{-4}$ , the girth-4 graph is substantially better. We have no exact explanation for this, but we suspect it is due to some property of the second largest eigenvalues. For each of these graphs, there are 8 eigenvalues (counted with multiplicity) of modulus  $\sqrt{2}$ , but the number of *distinct* eigenvalues of modulus  $\sqrt{2}$  differs: the girth-4 graph has only two, namely  $\pm\sqrt{2}i$ , while the graph with two 2-cycles has 6 such and the graph with three 2-cycles has 4.

The preceding figures also show that the numerical precision used for implementing the SPA plays a strong role in the performance. In particular, at  $\epsilon = 10^{-4}$  there is a very identifiable error floor in several of the performance curves. This corroborates a phenomenon that we observed in our experimental results on  $(3, 6)$ -regular codes of lengths 282 and 1002 conducted for [14]. In the course of determining the performance of a number of codes, we collected all vectors that caused decoding failure. Subsequent testing revealed that the vast majority of these vectors could be successfully decoded by reducing the termination parameter  $\epsilon$ . This raises two important questions: To what extent is the error floor an artifact of numerical precision as opposed to non-convergence of the algorithm (as is the case in the above examples)? Is there a method for creating graphs that are relatively resistant to numerical imprecision?

#### ACKNOWLEDGEMENTS

Joshua Lampkins contributed to the research for this article. In particular, he noticed the impact on the convergence of the sum-product algorithm when the graph  $G$  is bipartite.



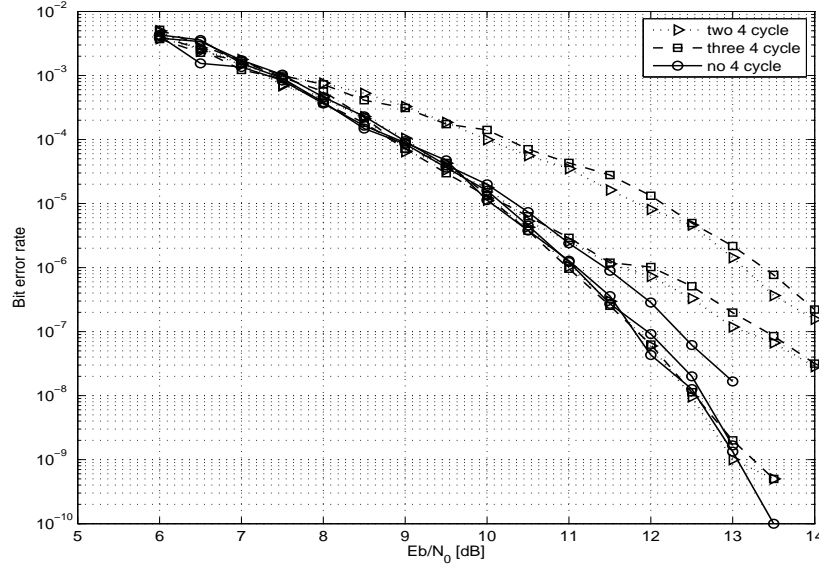


FIGURE 8. The performance of three 3-covers of  $K_{32}$  for  $\epsilon = 10^{-2}$ ,  $10^{-4}$ ,  $10^{-8}$ .

#### REFERENCES

- [1] Lowell W. Beineke and Robin J. Wilson, editors. *Selected topics in graph theory*. Academic Press Inc. [Harcourt Brace Jovanovich Publishers], London, 1978.
- [2] S.K. Chilappagari, S. Sankaranarayanan, and B. Vasic. Error floors of ldpc codes on the binary symmetric channel. In *IEEE International Conference on Communications (ICC '06)*, volume 3, pages 1089–1094, June 2006.
- [3] Seifollah Louis Hakimi and Jon G. Bredeson. Graph theoretic error-correcting codes. *IEEE Trans. Information Theory*, IT-14:584–591, 1968.
- [4] Yang Han and W.E. Ryan. Ldpc decoder strategies for achieving low error floors. In *Information Theory and Applications Workshop, 2008*, pages 277–286, Jan. 2008.
- [5] Frank Harary. *Graph theory*. Addison-Wesley Publishing Co., Reading, Mass.-Menlo Park, Calif.-London, 1969.
- [6] Roger A. Horn and Charles R. Johnson. *Matrix analysis*. Cambridge University Press, Cambridge, 1985.
- [7] M. Ivkovic, S.K. Chilappagari, and B. Vasic. Eliminating trapping sets in low-density parity-check codes by using Tanner graph covers. *IEEE Transaction on Information Theory*, 54(8):3763–3768, Aug. 2008.
- [8] Ralf Koetter, Wen-Ching W. Li, Pascal O. Vontobel, and Judy L. Walker. Pseudo-codewords of cycle codes via zeta functions. In *Proceedings IEEE Inform. Theory Workshop*, pages 7–12, San Antonio, TX, USA, 2004. IEEE.
- [9] Ralf Koetter, Wen-Ching W. Li, Pascal O. Vontobel, and Judy L. Walker. Characterizations of pseudo-codewords of (low-density) parity-check codes. *Adv. Math.*, 213(1):205–229, 2007.
- [10] S. Lampoudi, J. Brevik, and M. E. O’Sullivan. Combinatorial properties as predictors for the performance of the sum-product algorithm. preprint, 2010.
- [11] D. J. C. MacKay. Good error-correcting codes based on very sparse matrices. *IEEE Trans. Inform. Theory*, 45(2):399–431, 1999.
- [12] David J. C. MacKay and M. J. Postol. Weaknesses of Margulis and Ramanujan–Margulis low-density parity-check codes. In *Proceedings of MFCSIT2002, Galway*, volume 74 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 2003.
- [13] Henryk Minc. *Nonnegative matrices*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons Inc., New York, 1988. A Wiley-Interscience Publication.

- [14] M. E. O'Sullivan, J. Brevik, and R. Wolski. The performance of LDPC codes with large girth. In *Proc. of the 43-th Annual Allerton Conference on Communication, Control, and Computing*, Sept. 2005.
- [15] S.K. Planjery, S.K. Chilappagari, B. Vasic, D. Declercq, and L. Danjean. Iterative decoding beyond belief propagation. In *Information Theory and Applications Workshop (ITA), 2010*, pages 1 –10, Jan. 2010.
- [16] S.K. Planjery, D. Declercq, S.K. Chilappagari, and B. Vasic and. Multilevel decoders surpassing belief propagation on the binary symmetric channel. In *Proceedings, 2010 IEEE International Symposium on Information Theory*, pages 769 –773, June 2010.
- [17] T. Richardson. Error floors of LDPC codes. In *Proc. of the 42-nd Annual Allerton Conference on Communication, Control, and Computing*, pages 1426–1435, 2004.
- [18] T. Richardson, A. Shokrollahi, and R. Urbanke. Design of capacity-approaching irregular low-density parity-check codes. *IEEE Trans. Inform. Theory*, 47(2):619–639, 2001.
- [19] Tom Richardson and Rüdiger Urbanke. *Modern Coding Theory*. Cambridge University Press, 2008.
- [20] H. M. Stark and A. A. Terras. Zeta functions of finite graphs and coverings. *Adv. Math.*, 121(1):124–165, 1996.
- [21] Jean-Pierre Tillich and Gilles Zémor. Optimal cycle codes constructed from Ramanujan graphs. *SIAM J. Discrete Math.*, 10(3):447–459, 1997.
- [22] B. Vasic, S.K. Chilappagari, D.V. Nguyen, and S.K. Planjery. Trapping set ontology. In *47th Annual Allerton Conference on Communication, Control, and Computing, 2009*, pages 1 –7, Sept. 2009.

DEPARTMENT OF MATHEMATICS AND STATISTICS, LONG BEACH STATE UNIVERSITY  
*E-mail address:* `jbrevik@csulb.edu`

DEPARTMENT OF MATHEMATICS AND STATISTICS, SAN DIEGO STATE UNIVERSITY  
*E-mail address:* `mosullivan@mail.sdsu.edu`